



Jmix REST Data Store.

Microservices in one click

About



Cherkasov Dmitry

Java Backend dev

Jmix DevRel



Activities:

- Java /JVM
- Cloud / Microservices
- *Jmix Framework* developer
- *Different PL*
- *Asio & guitars*



One dev team sad story



Boilerplate communication

- DTO
- Mapper
- Controller
- ACL



*More code –
More work*

Spring Boot – solution?

- ✓ Mapper
- ✓ Controller
- ✓ Docs
- ✗ DTO (Projection)
- ✗ ACL
- ✗ More Repository
- ✗ More Queries

Spring Boot
Rest Repository



```
@RepositoryRestResource
public interface CategoryRepo extends JpaRepository<Category, Integer> {

    @RestResource(exported = false)
    @Query("select c as category, count(p) as quantity from Category c left join Product p on c.id = p.categoryId")
    CategoryProjection getDto(Integer categoryId)

    @RestResource(exported = false)
    @Query("select c as category, count(p) as quantity from Category c left join Product p on c.id = p.categoryId")
    List<CategoryProjection> getDtos();

    @RestResource(exported = false)
    @Query("select c as category, count(p) as quantity from Category c left join Product p on c.id = p.categoryId")
    Page<CategoryProjection> getDtos(Pageable pageable)
}
```

```
@Relation(value = "category", collectionRelation = "products")
public class CategoryDto implements CategoryProjection {

    private final Category category;
    private final Long quantity;

    // skipped
}
```

Jmix solution!

- Meta-data
 - ACL x Data
 - Fetch Plan
 - Reflection
-
- ✓ DTO
 - ✓ Mapper
 - ✓ Controller
 - ✓ ACL
 - ✓ Docs

Jmix Generic REST



THE END

Проблема в клиенте?

- OpenAPI: Gen-d Client
- Schema: DTO
- Production

Boilerplate
Integration
Call

```
@Install(to = "restaurantsDL", target = Target.DATA_LOADER)  ± kartondev
private List<RestaurantDTO> restaurantsLoaderLoadDelegate(final LoadCor
AppUser appUser = (AppUser) currentAuthentication.getUser();
return restaurantClient.listRestaurants(appUser.getUserToken());
}
```

```
private void initRestaurantDc(String restaurantId) { 1 usage  ± kartondev *
AppUser appUser = (AppUser) currentAuthentication.getUser();

restaurantDL.setLoadDelegate(e -> restaurantClient.getRestaurantById
restaurantDL.load();
}
```


Проблема в клиенте!

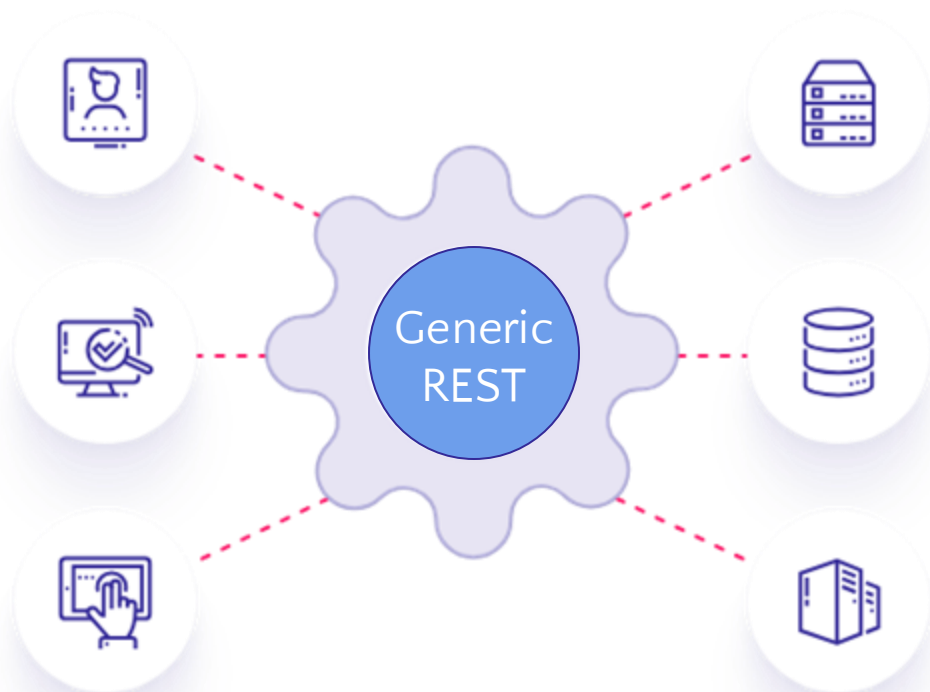
- Client – everywhere
- Versioning
- **Boilerplate**



Ну и так сойдет, никто не умер

But the Jmix!

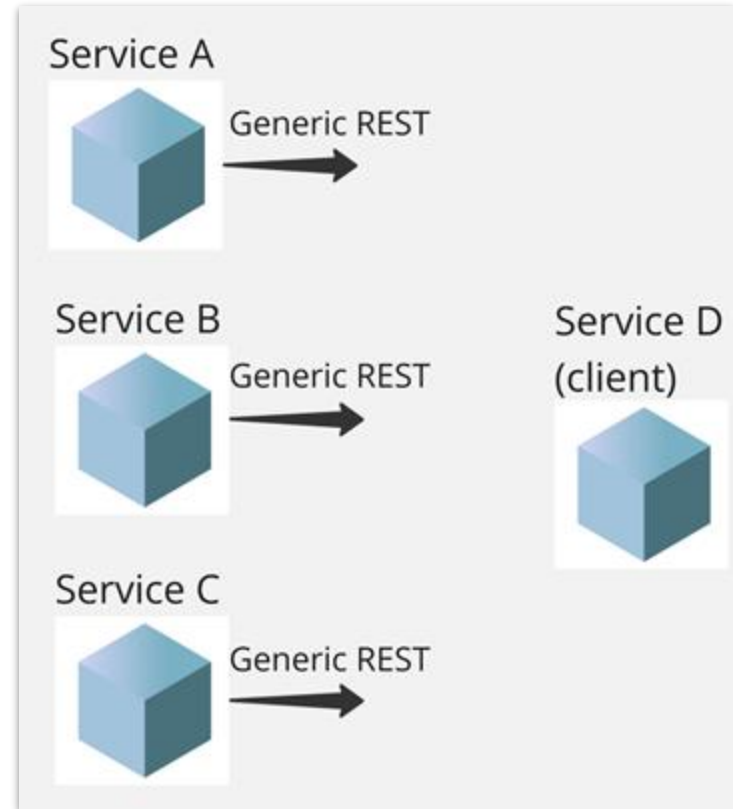
- Meta-data
- ACL x Data
- Fetch Plan
- Reflection
- **Generic REST**



Retrospective

- Meta-data
- Fetch Plan
- Generic REST – **format**

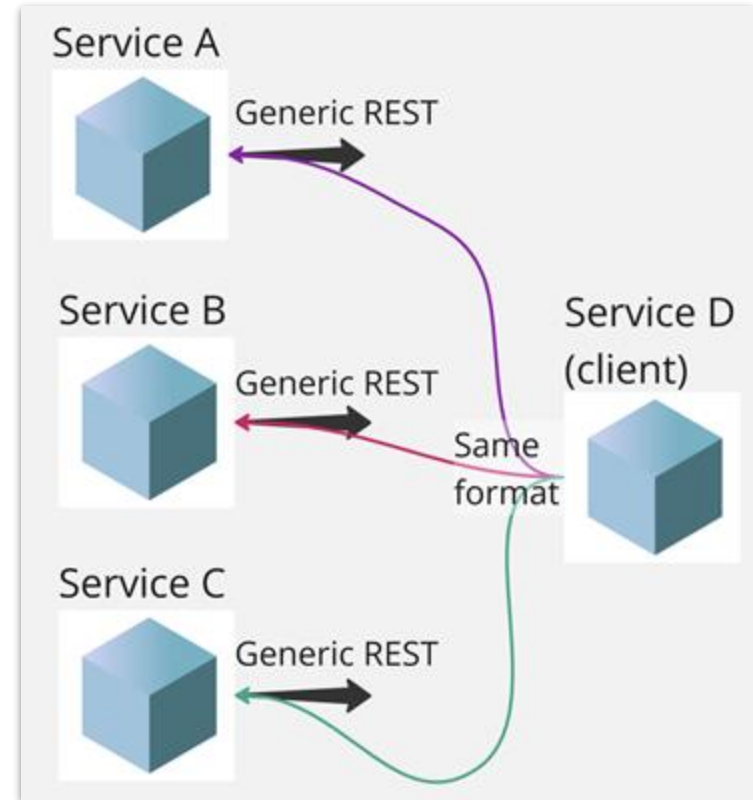
Different **Entity**
Same **API**



Retrospective

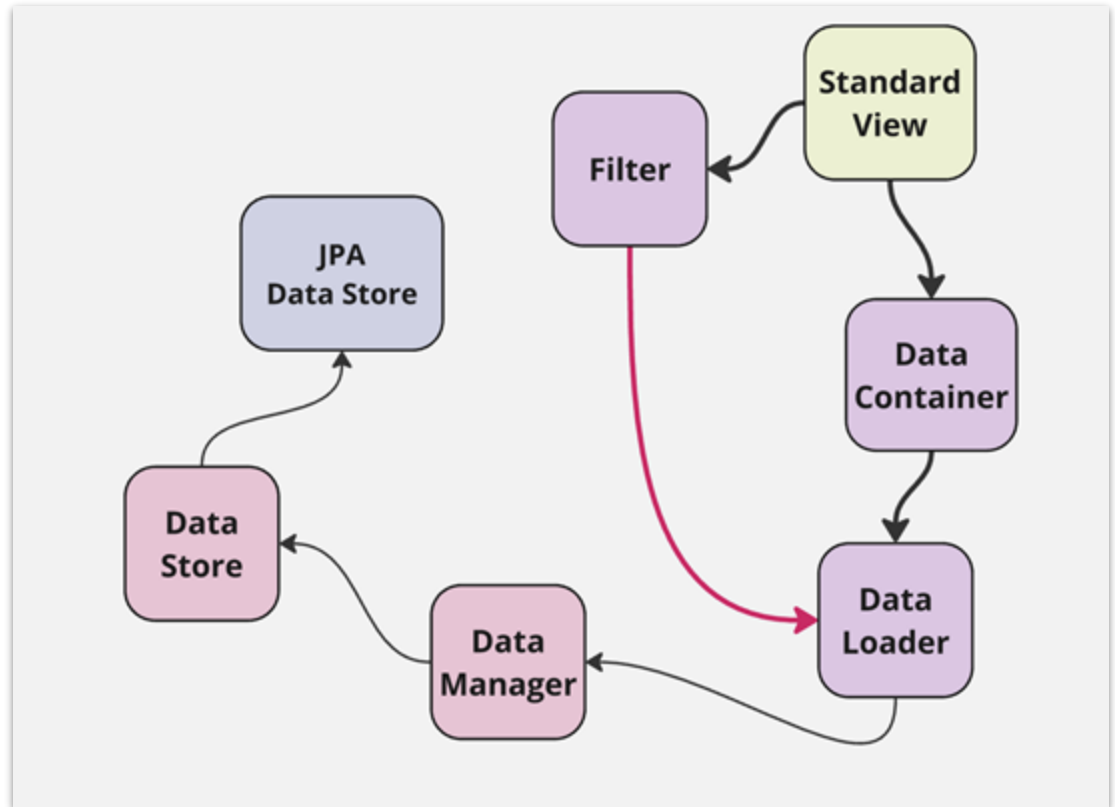
- Meta-data
- Fetch Plan
- Generic REST – **format**

Different Entity
Same **Request**



Jmix View Data – magic

1. View
 - a. Filter
 - b. etc
2. Container
3. Loader
4. Manager
5. Store
6. JPA Store



Data Store



1. DS = database
2. DS – many
3. DS > JPA
4. DS => **Data API**

```
public interface DataStore { no usages 4 implementations
    String getName(); 3 implementations

    void setName(String name); 3 implementations

    @Nullable 2 implementations
    Object load(LoadContext<?> context);

    List<Object> loadList(LoadContext<?> context); 2 implementations

    long getCount(LoadContext<?> context); 2 implementations

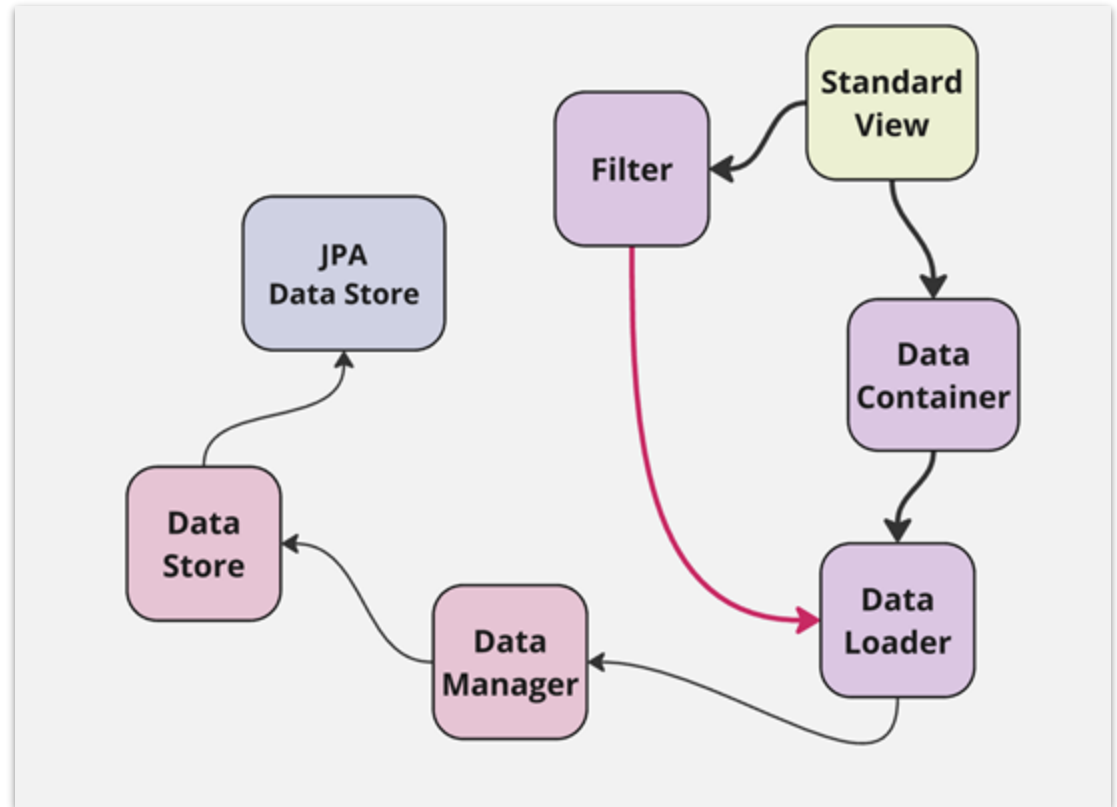
    Set<?> save(SaveContext context); 3 implementations

    List<KeyValueEntity> loadValues(ValueLoadContext context); 2 implementations

    long getCount(ValueLoadContext context); 2 implementations
}
```

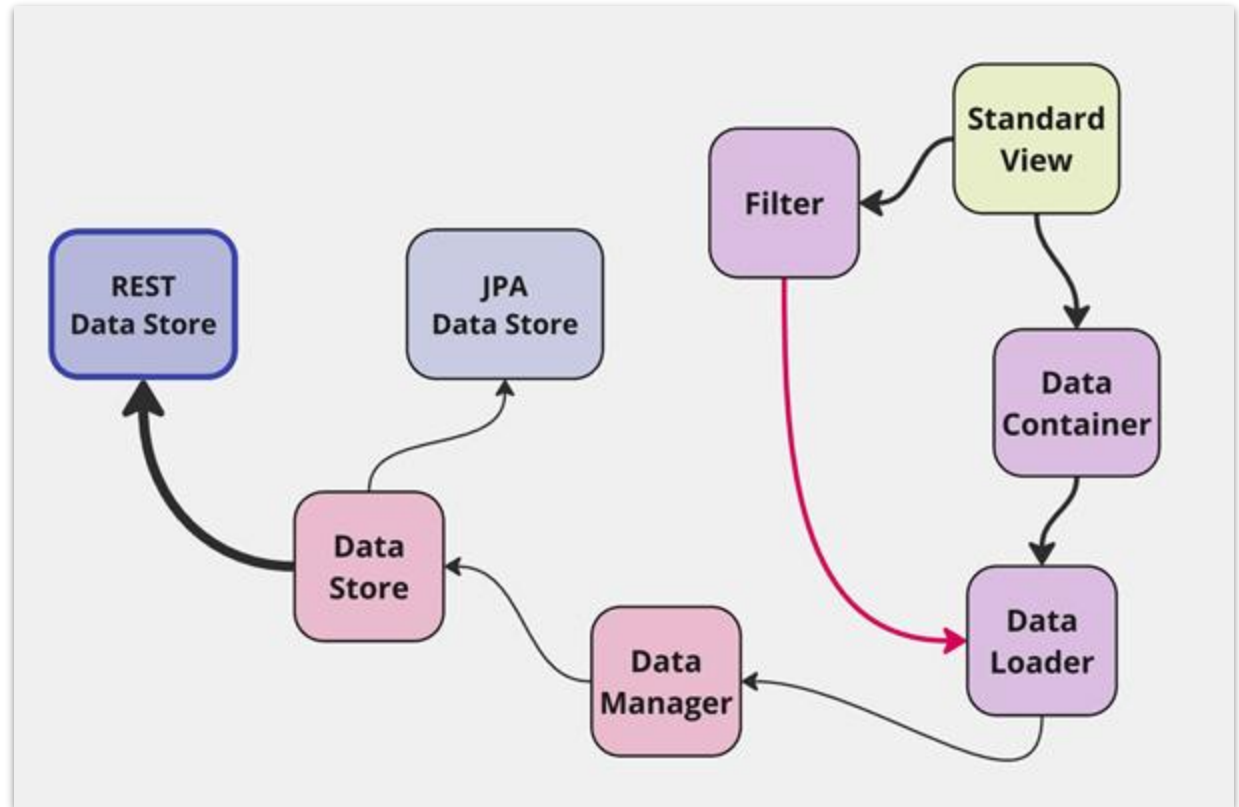
Jmix View Data – magic

8. > ???



Jmix View Data – magic

8. REST DataStore



Data Store



Complexity:

- Auth
- Constraint & ACL
- Fetch Plan
- Unloaded attr
- Jmix API
 - Listeners
 -
- Collisions
-

```
public interface DataStore { no usages 4 implementations
    String getName(); 3 implementations

    void setName(String name); 3 implementations

    @Nullable 2 implementations
    Object load(LoadContext<?> context);

    List<Object> loadList(LoadContext<?> context); 2 implementations

    long getCount(LoadContext<?> context); 2 implementations

    Set<?> save(SaveContext context); 3 implementations

    List<KeyValueEntity> loadValues(ValueLoadContext context); 2 implementations

    long getCount(ValueLoadContext context); 2 implementations
}
```

Functional requirements

Requirements:

- Service name
- DTO Projection
- Entity Graph

REST DataStore:

- Service name
- DTO (Jmix Entity)
- Fetch plan (*named*)

REST Data Store



Configuration

```
jmix.core.additional-stores = serviceapp  
jmix.core.store-descriptor-serviceapp = restds_RestDataStoreDescriptor
```

```
serviceapp.baseUrl = http://localhost:8081  
serviceapp.clientId = clientapp  
serviceapp.clientSecret = clientapp123
```

```
serviceapp.authenticator = restds_RestPasswordAuthenticator  
jmix.restds.authentication-provider-store = serviceapp
```

```
@JmixEntity
@Table(name = "REGION")
@Entity
public class Region {
    @JmixGeneratedValue
    @Column(name = "ID", nullable = false)
    @Id
    private UUID id;

    @Column(name = "VERSION", nullable = false)
    @Version
    private Integer version;

    @InstanceName
    @Column(name = "NAME", nullable = false)
    @NotNull
    private String name;

    // getters and setters
}
```

Serviceapp

Service ref

ClientApp

Name ref

```
@Store(name = "serviceapp")
@JmixEntity
public class Region {
    @JmixGeneratedValue
    @JmixId
    private UUID id;

    private Integer version;

    @InstanceName
    @NotNull
    private String name;

    // getters and setters
}
```

```
@JmixEntity
@Table(name = "REGION")
@Entity
public class Region {
    @JmixGeneratedValue
    @Column(name = "ID", nullable = false)
    @Id
    private UUID id;
```

```
    @Column(name = "VERSION")
    @Version
    private Integer version;
```

```
    @InstanceName
    @Column(name = "NAME", nullable = false)
    @NotNull
    private String name;
```

// getters and setters

Serviceapp

Service ref

```
@Store(name = "serviceapp")
@JmixEntity
public class Region {
    @JmixGeneratedValue
    @Id
    private UUID id;
```

```
@Store(name = "serviceapp")
@JmixEntity
@RestDataStoreEntity(remoteName = "Region")
public class RegionDto {
    // ...
}
```

Name ref

```
private String name;
```

// getters and setters

Data Manager

```
public List<User> loadByCondition(String lastName) { no usages
    return dataManager.load(User.class) FluentLoader<User>
        .condition(PropertyCondition.equal(property: "lastName", lastName)) ByCondition<User>
        .list();
}
```


```
{
  "conditions": [
    {
      "property": "lastName",
      "operator": "=",
      "value": "$$youLastName$$"
    }
  ]
}
```

Data Manager

```
List<User> loadByQuery(String lastName) { no usages
    String query = ""
    {
        "property": "lastName",
        "operator": "=",
        "value": "%s"
    }""
    .formatted(lastName);

    return dataManager.load(User.class) FluentLoader<User>
        .query(query) ByQuery<User>
        .list();
}
```


Data Manager

 Jmix Documentation

Version 2 ▾

JMX Console

Kanban

LDAP

Maps

Multitenancy

Notifications

OpenID Connect

Pivot Table

Pessimistic Locking

Quartz

Reports

REST API

Getting Started with REST

Access Control

Entities API

Load Entities

Create Entities

Update Entities

Delete Entities

Files API

Messages API

Metadata API

User Session API

Business Logic

CORS

API Documentation

REST Properties

REST DataStore

Search

Superset

UI Constraints

Jmix Documentation / Add-ons / REST API / Entities API / Load Entities

value

the value to search for. Value is not required for the `notEmpty` and `isNull` operators.

Additionally, conditions can be combined via `AND`, `OR` group conditions to define a more complex filter criterion. The JSON structure of the filter definitions looks like this:

Filter Criterion JSON structure

```
{
  "conditions": [
    {
      "group": "OR",
      "conditions": [
        {
          "property": "stringField",
          "operator": "=",
          "value": "stringValue"
        },
        {
          "property": "intField",
          "operator": ">",
          "value": 100
        }
      ]
    },
    {
      "property": "booleanField",
      "operator": "=",
      "value": true
    }
  ]
}
```

This is a representation of the Filter criterion: `((stringField = stringValue) OR (intField > 100) AND (booleanField = true))`.

```
List<User> loadByQuery(String lastName) { no usages
    String query =
    {
        "conditions": [
            {
                "group": "OR",
                "conditions": [
                    {
                        "property": "stringField",
                        "operator": "=",
                        "value": "%stringValue"
                    },
                    {
                        "property": "intField",
                        "operator": ">",
                        "value": 100
                    }
                ]
            },
            {
                "property": "booleanField",
                "operator": "=",
                "value": true
            }
        ]
    }

    .formatted(lastName);

    return dataManager.load(User.class) FluentLoader<User>
        .query(query) ByQuery<User>
        .list();
}
```

Data Manager

Query builder

Under the hood

- Query string
- Condition
- Sub-condition string
- Complex condition

```
39  @Com LogicalCondition condition = LogicalCondition.and(  
40  String result = builder.build( query: ""  
41  
42      {  
43          "property": "name",  
44          "operator": "=",  
45          "value": "alpha"  
46      }""");  
47  
48  assertThatJsonEquals(result, expected: ""  
49  
50      {  
51          "conditions": [  
52              {  
53                  "property": "lastName",  
54                  "operator": "=",  
55                  "value": "$$youLastName$$"  
56              }  
57          ]  
58      }""");
```

Use cases

- Data Manager / Service
-

```
@Override no usages
@Transactional
public User createUser(String username, String email) {
    User user = dataManager.create(User.class);
    user.setUsername(username);
    user.setEmail(email);
    return dataManager.save(user);
}

@Override 2 usages
public User getUserById(UUID id) {
    return dataManager.load(User.class) FluentLoader<User>
        .id(id) ById<User>
        .one();
}

@Override no usages
public List<User> getAllUsers() {
    return dataManager.load(User.class) FluentLoader<User>
        .all() ByCondition<User>
        .list();
}

@Override no usages
@Transactional
public User updateUser(UUID id, String username, String email) {
    User user = getUserById(id);
    if (user != null) {
        user.setUsername(username);
        user.setEmail(email);
        return dataManager.save(user);
    } else {
        throw new IllegalArgumentException("User not found with id: " + id);
    }
}

@Override no usages
@Transactional
public void deleteUser(UUID id) {
    User user = getUserById(id);
    if (user != null) {
        dataManager.remove(user);
    } else {
        throw new IllegalArgumentException("User not found with id: " + id);
    }
}
```

Use cases

- Data Manager / **Service**
- Data Manager / **Ui Controller**
-

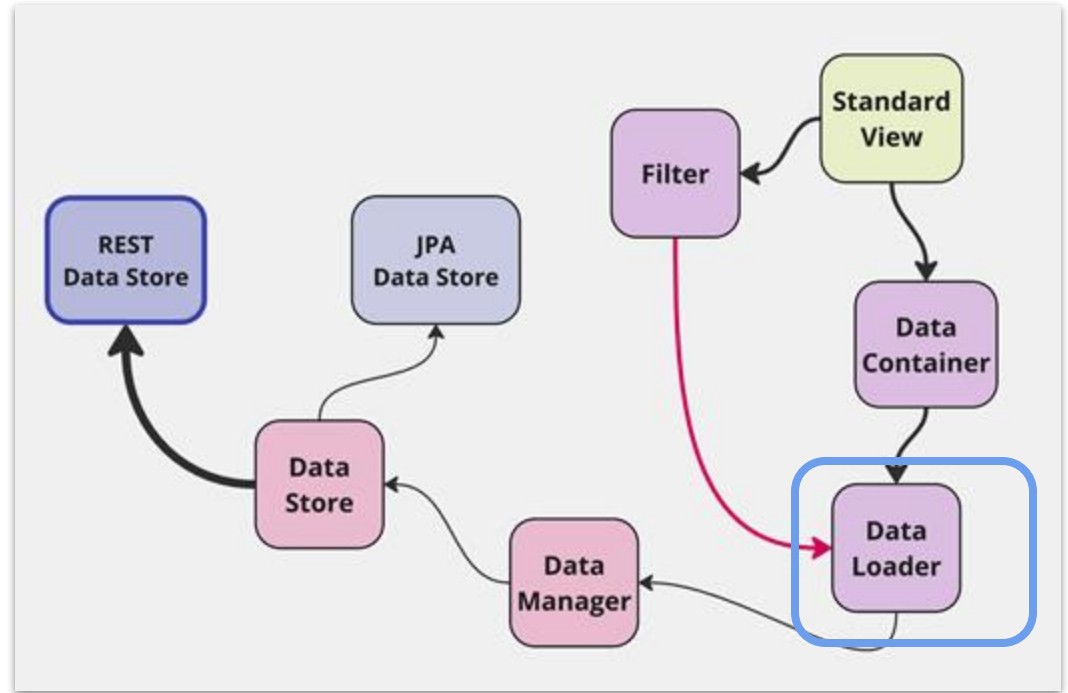
```
@ViewComponent 1 usage
private CollectionContainer<User> usersDc;
@Autowired
private DataManager userManager;

@Subscribe
public void onBeforeShow(final BeforeShowEvent event) {
    usersDc.setItems(loadByCondition( lastName: "Smith"));
}

public List<User> loadByCondition(String lastName) { 1 usage
    return userManager.load(User.class) FluentLoader<User>
        .condition(PropertyCondition.equal( property: "lastName", lastName)) ByCondition<User>
        .list();
}
```

Use cases

- Data Manager / **Service**
- Data Manager / **View**
-



Use cases

- Data Manager / **Service**
- Data Manager / **View**
- View / **Data Loader**
- ...

```
<collection id="customersDc" class="com.company.clientapp.entity.Customer">
  <fetchPlan extends="_base" />
  <loader id="customersDl" readOnly="true">
    <query>
      <![CDATA[
        {
          "property": "region",
          "operator": "=",
          "parameterName": "region"
        }
      ]]>
    </query>
  </loader>
</collection>
```

Use cases

- Data Manager / **Service**
- Data Manager / **View**
- View / **Data Loader**
- View / **ItemContainer**
- *etc...*

```
<entityComboBox id="regionField" property="region">
  <itemsQuery class="com.company.clientapp.entity.Region"
    searchStringFormat="${inputString}">
    <fetchPlan extends="_base"/>
    <query>
      <![CDATA[
        {
          "property": "name",
          "operator": "contains",
          "parameterName": "searchString"
        }
      ]]>
    </query>
  </itemsQuery>
</entityComboBox>
```

Demo – Pet Clinic



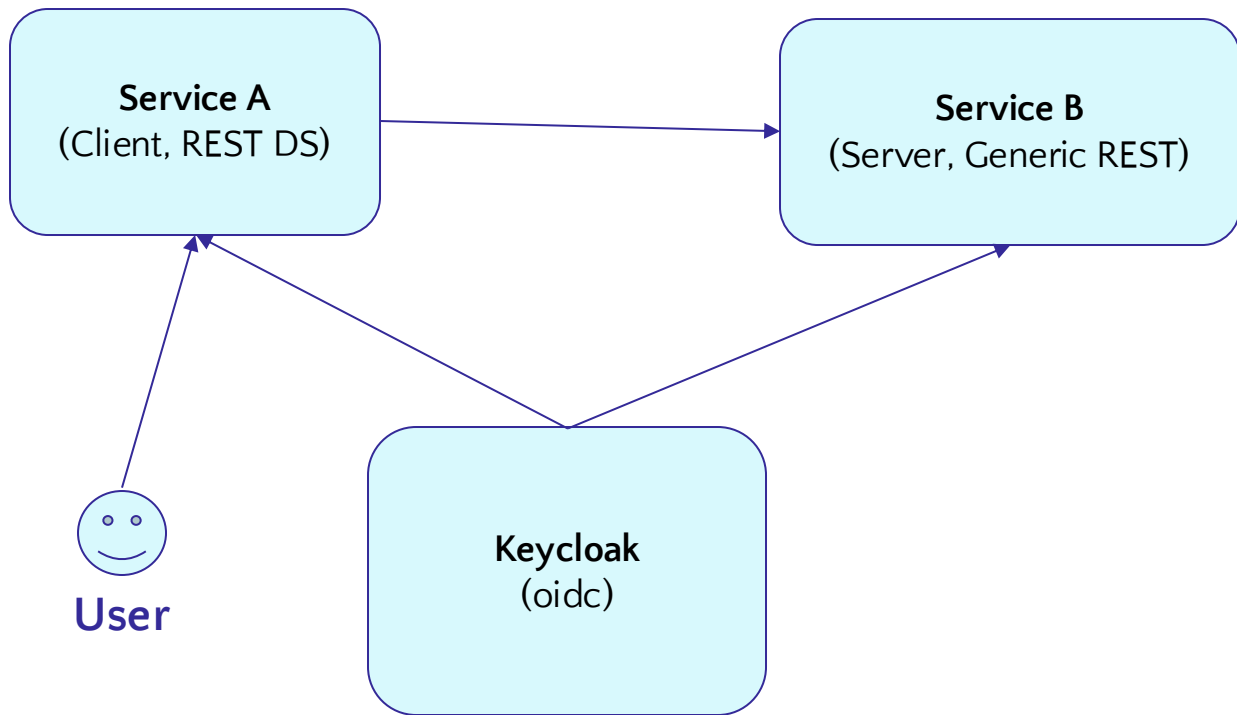
Application cases:

Cases & BL:

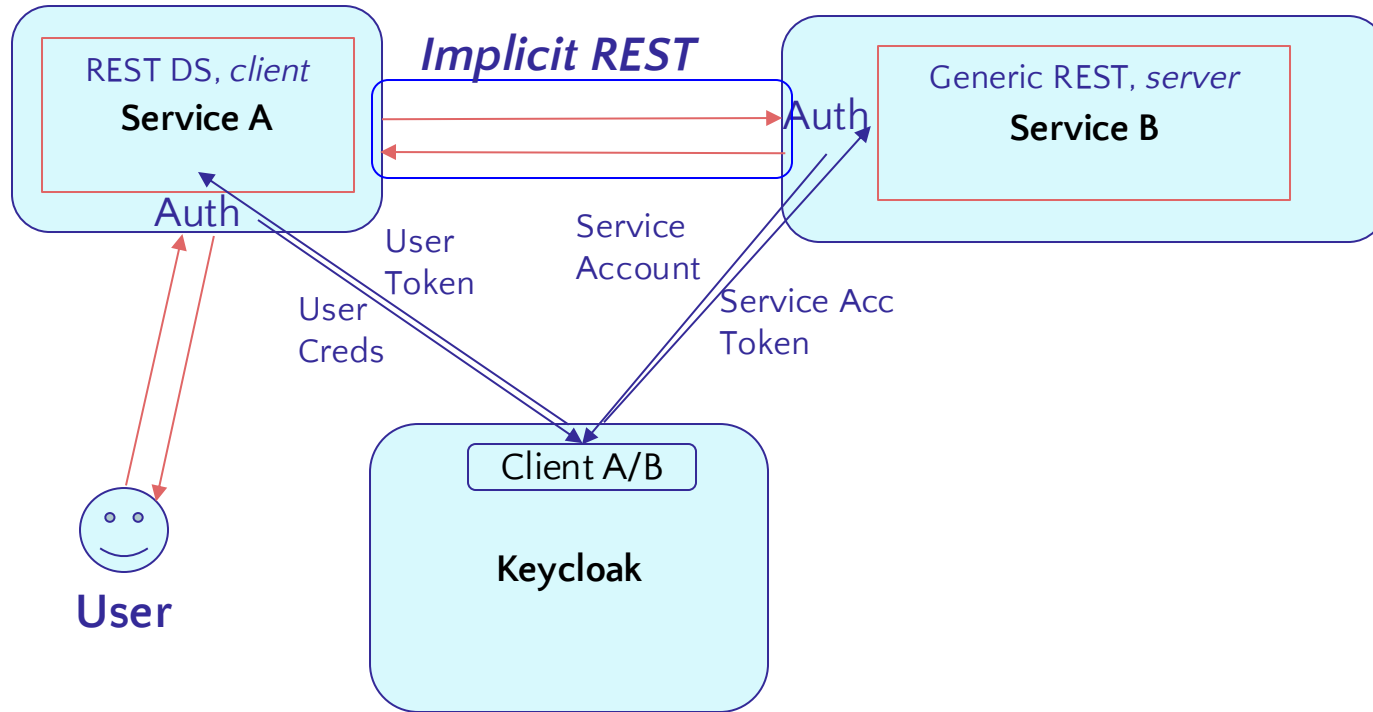
- Owner Service
- Pet Service
- Veterinarian Service



Tech case



Tech case – Implicit REST



Microservices



Template

Full-Stack Application (Java)

Full-Stack Application (Kotlin)

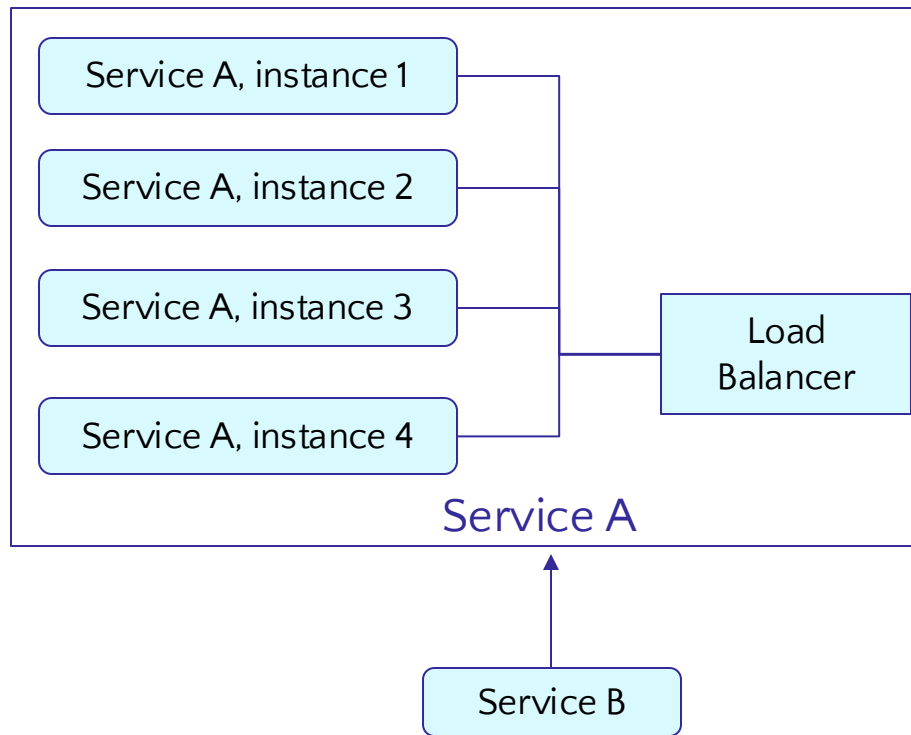
Add-On (Java)

Add-On (Kotlin)

REST Service Application

Load Balancer

- Spring Cloud
- Nginx
- K8s
-



Service Discovery

- Spring Cloud Eureka
- Spring Cloud Netflix
- K8s
-

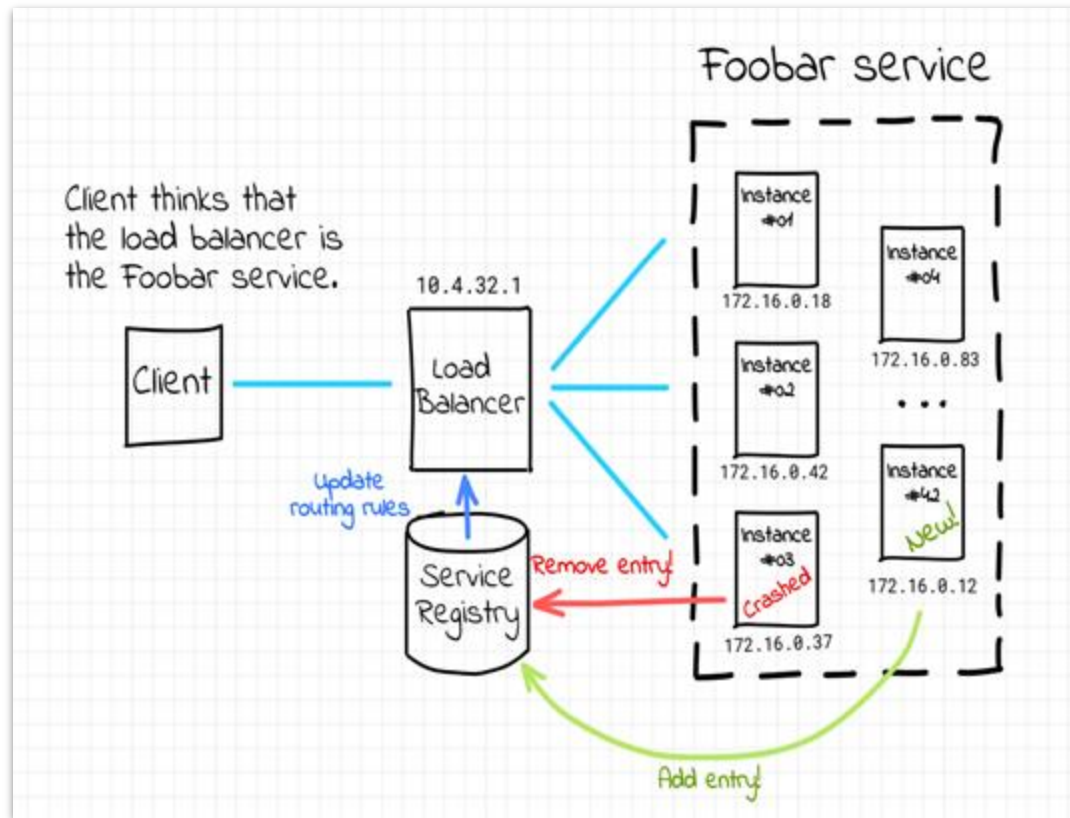


@DynamicPropertySource

```
petservice.baseUrl = http://localhost:8080  
petservice.clientId = petclinic  
petservice.clientSecret = cPNDGCEL873ES9LR0f8ktU1qbmp0PbBg  
petservice.authenticator = petclinic_OidcRestClientCredentialsAuthenticator
```

Service Discovery

- Eureka
- Netflix
- K8s
-



Cloud Config

- Spring Cloud Config
- K8s ConfigMap
- Vault
-



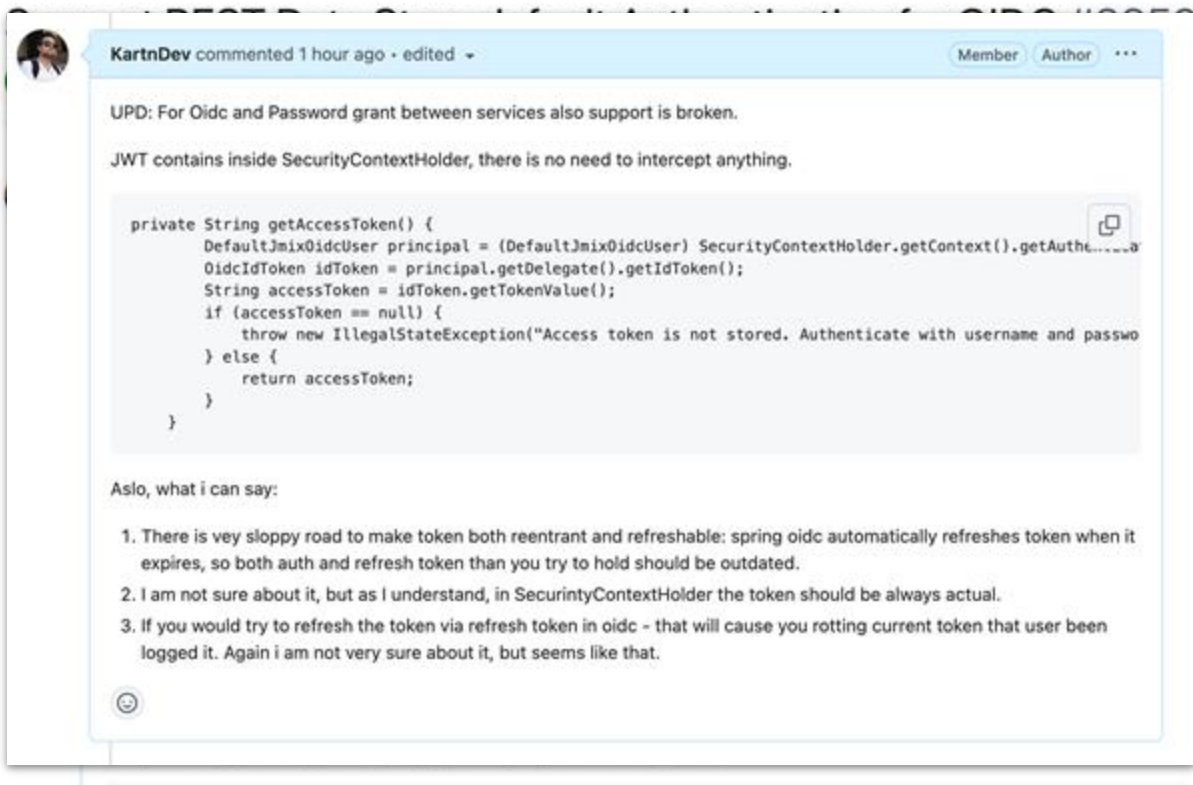
Future of REST Data Store

- Fetch Plan
- Direct **instance & collection** refs
- Inverse **instance & collection** refs
- R&D
- Implicit REST
- Easy connect
- Fetch plan & metadata = schema
-



WIP & flow

- OIDC
- Grant Flow
 - Service Acc
 - User Token
 - Impersonation
- Fetch plan
- Bugs



Result

- Sweeping **CRUD** under the carpet
- Implicit REST
- Secured
- **Develop speed**





Thanks for
your attention



/KartnDev/jmix-petclinic-ms